This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2934890, IEEE Access

A Hybrid Task Scheduling Scheme for Heterogeneous Vehicular Edge Systems

Xiao Chen, *Member, IEEE,* Nigel Thomas, Tianming Zhan and Jie Ding

Abstract—Enhanced wireless communication improves the connectivity of vehicular networks in which vehicles are utilized as infrastructures for communication and computation. Thus, a new concept "Vehicular Edge Computing (VEC)" is formed. As VEC utilizes a collaborative multitude of near-user edge resources (i.e. vehicles) in the Internet of Vehicles, the capability of these joint resources becomes heterogeneous especially in their movements. Therefore, one critical problem is how to efficiently schedule each task under such mobile environments. For the reason, we propose a hybrid dynamic scheduling scheme (HDSS) that has the ability to optimize the task scheduling dynamically based on the changeable system environments. HDSS provides a decision function (DF) to select a better-performed scheduling algorithm from two provided candidates: the queue-based dynamic scheduling (QDS) algorithm and the time-based dynamic scheduling (TDS). QDS coincides with the Join-the-Shortest Queue scheme, which decides the scheduling by sorting out a server with the shortest queue-length; nevertheless, TDS is novel scheme that is designed to implement task allocation by estimating the waiting time of each server in order to select a server with the fastest response. Finally, this research generates formal models of each scheduling algorithm and the hybrid scheduling scheme to conduct performance evaluation with a fluid flow approximation technique. The analysis results in a superior performance of HDSS in the unstable VEC environments.

Index Terms—Internet of Vehicles, Edge Computing, Scheduling Algorithm, Formal Modelling.

1 Introduction

Internet of vehicles (IoV) is considered a crucial part of intelligent transportation systems (ITS), as they can support a rich set of mobile services from large-scale data sharing to comprehensive computing services [1]. IoV systems are able to take on more service demands, such as complex computation, large-scale storage and high-level data dissemination. Therefore, to meet these demands, new technologies are developed but still have limitations. Traditional central cloud services generate extra latency due to frequent upstream/downstream transmissions [2]; future cellular networks provide high data rate and capacity but are still limited in spectrum resources [3]; RSUs enlarge communication capacity of vehicular networks, however their full area deployment requires extreme-high construction cost [4]. Mobile cloud computing supports powerful computational capability but is also costly and time consuming especially for real-time computation demand due to its client-server communication architecture [5]; and requires extra virtual resource management supported by a high quality of network connections and road-side infrastructures [6]. Therefore, it remains a great challenge for researchers to explore an efficient solution to deal with communication and computation demands of IoV systems [7], [8].

Nowadays, a growing number of vehicles will suffer from a heavy traffic with slow-moving speed especially in an urban environment. If most future vehicles are fully equipped with smart systems and networking devices, they will be considered underutilized resources. Thus, full use of these resources will greatly expand the physical capability of IoV systems. Thus, the new concept of "Vehicular Edge Computing (VEC)" is formed. Specifically, VEC is an architecture that wirelessly connects a collaborative multitude of on-board equipments to conduct a substantial amount of communication and computation [9]. VEC benefits data transmission, computing, storage and other application services due to its new features, such as the proximity to end users, dense geographical distribution and support for mobility [10]. Based on these advantages, VEC is considered for various applications, such as vehicular crowdsourcing applications [11] and delay-sensitive applications for smart cities [12]. However, as VEC joints various distributed resources, these resources are not always standing still, but sometimes keeping movements. In VEC systems, idle vehicular resources can be logically used by a set of virtual servers that are reached by nearby users. In this way, the users can offload their tasks from a root vehicle to these virtual servers and retrieve the computational results with low network latency. As the virtual edge servers are usually composed of geographically distributed resources (e.g., slow moving and parked vehicles nearby), a fundamental and crucial problem is where to offload a task from a root vehicle. This is a scheduling problem. There were some researches of scheduling in different views, such as security-based scheduling algorithm [13], fault recovery aware task scheduling [14], energy-efficient scheduling [15] and data-parallel frameworks [16]. However, most of investigated studies assume the scheduling operation in a homogeneous

- *X. Chen was jointly with State Key Laboratory of Software Development Environment at Beihang University, Beijing, China; and is with School of Informatics, University of Edinburgh, Edinburgh, UK, EH8 9AB. Email: xiao.chen@ed.ac.uk*
- *N. Thomas is with School of Computing Science, Newcastle University, Newcastle Upon Tyne, UK. Email: nigel.thomas@ncl.ac.uk*
- *T. Zhan is with School of Information Engineering, Nanjing Audit University, Nanjing, 211815, China. Email: ztm@nau.edu.cn*
- *J. Ding is with China Institute of FTZ Supply Chain, Shanghai Maritime University, Shanghai, China. Email: jding@shmtu.edu.cn*

Manuscript received April 19, 2005; revised August 26, 2015.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2934890, IEEE Access

system environment. Actually, in a real-world edge system, both server capability and job stream usually exist in a changeable situation, which can be represented by a stochastic process with a varying mean. For example, job streams usually reach the peak value in rushing hours and then drops to the lowest rate; on the other hand, virtual servers also have unstable capability due to moveable resources especially in a VEC system. Thus, there are two crucial issues that are: how to efficiently utilize these resources and how to guarantee the quality of service in such a mobile environment.

To address these issues, this paper mainly considers dynamic scheduling under the homogeneous VEC systems. The proposed hybrid dynamic scheduling scheme (HDSS) aims to adaptively select an optimal scheduling algorithm according to the current system conditions. In the HDSS, there are two candidate scheduling algorithms: a queue-based dynamic scheduling algorithm (QDS), which is designed by dynamically altering the scheduling process in terms of the transient queue length of servers coinciding with the traditional Join-the-Shortest-Queue scheme [17]; and a novel time-based dynamic scheduling algorithm (TDS), which decides the scheduling operation based on a prediction of server's response time rather than the length of waiting queue. To evaluate the performance of both algorithms, formal analysis are conducted by implementing two algorithms in VEC systems models with various conditions. Then, the architecture of HDSS is designed based on the analysis, which can improve the efficiency and adaptivity of the dynamic scheduling scheme. Further experiments reveal the superior performance of HDSS by comparing with a similar scheme that is mainly based on JSQ algorithm. To obtain transient performance measurements, a novel stochastic modelling technique, Performance Evaluation Process Algebra (PEPA), is applied due to its strengths in formality and compositionality, as well as a novel fluid-flow analysis by solving a set of ordinary differential equations (ODEs) derived from PEPA models.

Section 2 enhances the necessity and main contributions of our work by reviewing the related researches. Section 3 introduces a model framework of scheduling operation in a VEC system. Section 4 defines two dynamic scheduling algorithms. Section 5 presents the details of proposed HDSS. Section 6 demonstrates the modelling details of HDSS with two dynamic scheduling algorithms; thereafter, the fluid-flow analysis is conducted to evaluate the performance of two dynamic scheduling algorithms and HDSS in Section 7. Section 8 finally concludes the whole research and discusses a sustained research plan.

2 Related Work

Scheduling problem is still a research focus in distributed clustering computing area. For example, a general taxonomy is presented by Lopes [18] to demonstrate scheduling problems and related solutions in distributed systems. Over 100 recent scheduling problems and solutions are mentioned in the taxonomy.

During past years, cloud computing became popular due to its new form of Internet-based computing. To ensure efficient use of cloud services, a growing number of researches focus on scheduling of tasks, resources or workflows on a cloud platform. A stochastic model is considered by Maguluri [19] for a cloud computing cluster in which arriving jobs follow a stochastic process and request virtual machines (VMs). In the research, several widely used algorithms are explored by a stochastic model, such as MaxWeight algorithm, Best-Fit scheduling algorithm and Join-the-Shortest-Queue (JSQ) algorithm based on the evaluation of throughput and time delay of these alternative algorithms. Moreover, there are many other researches of scheduling problems in cloud computing, which have been summarized by Tsai [20]. Additionally, recent scheduling research turns to the edge of network as a result of mass data transmission between wirelessly connected mobile devices.

Growing data stream becomes a burden of network and central servers; thus, edge-devices are considered extended resources to provide computation and storage services. Thereafter, a growing number of researchers begin their study on the scheduling problems of such edge or fog computing environments. Tan [23] develops a general scheduling model to minimize response time in the condition that jobs are generated randomly from mobile devices and offloaded to edge servers with upload/download delays. Zeng's research [24] proposes an efficient task scheduling and resource management strategy to minimize task completion time through a mixed-integer non-linear programming solution. Additionally, Bittencourt [25] investigates scheduling algorithms of fog computing in the view of application performance that is effected by user mobility.

According to literature reviews, researchers, e.g., Feng [21], [22], explores a framework which aggregates vehicular edge resources to proved computation services, and then develops two decentralized solutions to achieve efficient resource sharing by using multiaccess networks. Moreover, some other researches investigate task scheduling (e.g., [23], [24], [25]) or resource allocation (e.g., [26], [27], [28]) issues for fog computing systems; furthermore, all these researches have limitations in discussing a homogeneous system conditions especially for the edge systems. Although Mahmood [29] mentioned the heterogeneous issues of vehicular networking resources and used edge-based caching to improve network performance, its solution is developed in the view of resource management rather than a scheduling approach. Nevertheless, Mukhopadhyay [30] mentioned a hybrid JSQ-based scheduling scheme that is designed for a heterogeneous processor-sharing systems. In this research, traditional JSQ algorithm is used together with other scheduling schemes to efficiently fit a heterogeneous system environment. Suppose in a VEC system with heterogeneous server capability, a server with shorter queue length may not generate faster response when its serving capability is occasionally reduced to a lower level due to the leaving of vehicular resources. Thus, a more accurate decision method should be based on the estimated response time for the current task rather. Hence, our research also considered the scheduling issue in a heterogeneous environment, but proposed a completely novel scheduling algorithm (TDS) that makes a scheduling decision on the basis of shortest response time rather than the queue length. Furthermore, a hybrid scheduling scheme using the proposed TDS is designed for heterogeneous VEC systems.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2934890, IEEE Access

The contributions can be highlighted: First, a novel time-based dynamic scheduling (TDS) algorithm is proposed for VEC systems with heterogeneous server capacity; Second, a hybrid dynamic scheduling scheme (HDSS) is designed to apply multiple scheduling algorithms to a more comprehensive system environment; Third, a novel formal framework is also developed to model a dynamic scheduling process by using decision functions in the models.

3 VFC Scheduling Model

This section introduces a sketch of scheduling model on a scenario of VEC systems. To clearly illustrate the formal definition, a set of major notations should be specified in Table 1.

TABLE 1
Notations in Formal VEC Model

Notations	Descriptions
J; S	Task set and server set.
j; s	Singular task and singular server.
$p(Q)$	Queue-based dynamic scheduling algorithm.
$p(T)$	Time-based dynamic scheduling algorithm.
$Var(r)$	A function representing system conditions.
$DSF(v_1, v_2)$	Decision function with inputs derived from $Var(r)$.

To facilitate the understanding of HDSS, the section utilzes a scenario of VEC systems as shown in Fig. 1. In such a VEC system, each virtual server (e.g., S_1 and S_2 in Fig. 1) usually comprises a set of mobile vehicular peers, and the mobility causes these heterogeneous conditions such as the varying virtual server resources and incoming-task streams. Unlike the cloud or cluster server systems supported by stable physical resources, VEC systems usually suffer from heterogeneous system conditions and result in negative effect on performance and service quality. These heterogeneous features require dynamic scheduling algorithms to adaptively and efficiently dispatch tasks to servers. The key issue is to design efficient dynamic special scheduling algorithms towards diverse system conditions such as varying arrivals and server capacity. To address the issue, this research aims to design a hybrid scheduling scheme with embedded novel or classic dynamic scheduling algorithms that can be selected by a decision support function through detecting system conditions.

To formally define the scheduling algorithms, a set of entities need be specified on the basis of notations in Table 1. First, each task in the system is specified as a 4-tuple $j_k =< k, v, \lambda, p >,\ j_k \in J\ (k = 1, 2, \cdots, n)$, in which k is the task ID, v is the index of a source vehicle, λ is the task generation rate, and p is the selected scheduling algorithm for a given task; Second, each server resource is represented as a 3-tuple $s_i =< i, v, \mu >,\ s_i \in S\ (i = 1, 2, \cdots, m)$, in which i is the server ID, v is the index of vehicle that provides physical resources, and μ is the capability of a given server. Fig. 1 depicts a general task scheduling procedure in VEC systems.

As shown in Fig. 1, vehicular service is supported by several sets of vehicles (e.g., S_1 and S_2) that are moving on the road or parking nearby. A source vehicle holds a series of tasks that is defined as the set J, and thereby each task

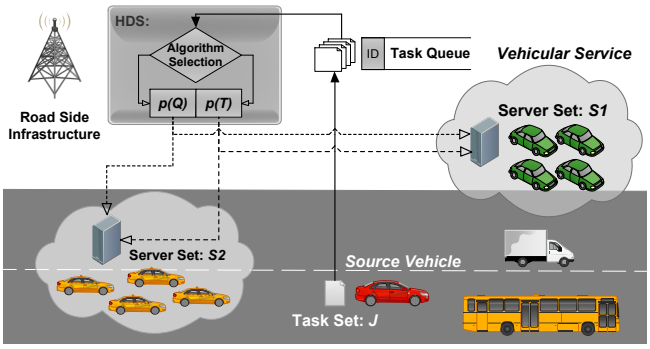

Fig. 1. VEC Scheduling Model

is sent to a queue of VEC monitor that manages vehicular servers in a local area. Next, queuing tasks are forwarded to a hybrid dynamic scheduler (HDS) to start a scheduling procedure. HDS first performs a selection of scheduling algorithm by invoking a decision-support function defined as $DSF(v_1, v_2)$ which accepts the outputs of function $Var(r)$ indicating the present system conditions. Thereafter, the result of $DSF(v_1, v_2)$ is used to determine which factor (i.e., either varying server capability or incoming-task streams) has a greater variation in the current system environment. Thereafter, a more appropriate scheduling algorithm (i.e., $p(Q)$ or $p(T)$) is confirmed by the decision-support function. Finally, with the confirmed scheduling algorithm, each task is dispatched to a selected server set (e.g., S_1 or S_2). Details of dynamic scheduling algorithms and decision-support function will be presented in the following section.

4 Scheduling Algorithms

In the multi-server system, scheduling process generally dispatches tasks to one of potential servers with a given constant probability, which is called "Static Scheduling". It is the simplest scheduling algorithm that is also named "Random Scheduling" in the condition of equivalent probability allocating to each server. Random scheduling is initially used for distributed system due to its low computation cost; however, with the development of computing technology, such simple random scheme might be deemed inapplicable to heterogeneous system environments, such as cloud or edge systems, as these systems make great demand on their performance (e.g., short queue length, fast response time and high throughput). For this reason, efficiency-aware scheduling scheme is required to ensure QoS.

This section will define a queue-based dynamic scheduling algorithm (QDS) and a novel time-based dynamic scheduling algorithm (TDS).

4.1 QDS: Queue-based Dynamic Scheduling

Owing to the consideration of heterogeneous systems, a classic scheduling algorithm (Join-the-Shortest-Queue, JSQ) is considered an appropriate option. JSQ is traditional but also popular in today's systems by specifying new scheduling mechanism based on JSQ (e.g., Ref. [30]). As such we extend the classic JSQ to a dynamic mode by giving its definition as follows.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2934890, IEEE Access

Definition 1. ***QDS Algorithm***
Queue-based Dynamic Scheduling (QDS) algorithm is defined for a scheduling process in which each task is scheduled to a target server with the shortest waiting queue at a given time point.

QDS algorithm is formally represented with $p(Q)$ that is composed of a decision function $DF_{QDS}(t)$ to make the scheduling decision. According to QDS algorithm's definition, $p(Q)$ can be formulated as:

$$p(Q) = C_q \cdot DF_{QDS}(t) = C_q \cdot \frac{1}{T[Q(t)]}, \tag{1}$$

where C_q is a constant coefficient that is set on account of system environment; t is a transient time point and $T[Q(t)]$ represents the transient queue-length at time t under QDS algorithm.

According to Eq.(1), QDS algorithm that is represented by $p(Q)$ is implemented by a decision function $DF_{QDS}(t)$, which can be computed in a scheduling process to decide the dynamic dispatch of tasks. As it follows the principle of JSQ algorithm and its computation can be easily achieved in operation, the formal proof is omitted in this section.

4.2 TDS: Time-based Dynamic Scheduling

With our investigation, JSQ-based scheduling scheme benefits a system that consists of distinct servers with different processing speeds. Nevertheless, today's heterogeneous systems, such as VEC systems, have not only distinct servers but also unstable capability at each server because these servers leverage changeable physical resources that are provided by moveable vehicles. In this situation, the number of queuing tasks is no longer the unique decisive factor of scheduling. In other words, a longer waiting queue might not mean the consistent longer waiting time due to the varying processing speed of servers. As a result, the JSQ-based scheduling algorithm may not make a right decision only based on the number of waiting tasks. For the reason, an alternative Time-based dynamic scheduling algorithm is presented and defined in this subsection.

Definition 2. ***TDS Algorithm***
Time-based Dynamic Scheduling (TDS) algorithm is defined for a scheduling process in which the scheduling decision is made on the basis of completion time of all queuing tasks of each server at a specified time point.

In the definition of TDS, $p(T)$ is used to represent the algorithm. Similarly, $p(T)$ can be represented with a decision function $DF_{TDS}(t)$ that is utilized to support the scheduling decision . The TDS algorithm, $p(T)$ can be formally represented as:

$$p(T) = C_T \cdot DF_{TDS}(t) = C_T \cdot \frac{E[R_s(K)]}{T[R(t)]}, \tag{2}$$

where C_T is a factor given by system environments and $E[R_s(K)]$ is the average response time of a server with K users; $T[R(t)]$ is the transient response time at time point t. From Eq.(2), the value of $E[R_sK)]$ is fixed, when $T[R(t)]$ increases which causes the decrease of $DF_{TDS}(t)$, tasks will be sent to the corresponding server with a slower rate. In other words, this means that less tasks will be scheduled to the server with increased $T[R(t)]$ based on the decision function $DF_{TDS}(t)$.

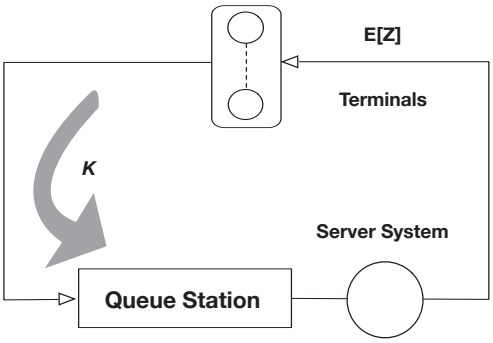

Fig. 2. $M|M|1||K$ Terminal Model

Nevertheless, in above Eq.(2), $E[R_s(K)]$ is an estimated average response time that is obtained from approximation rather than the accurately measured value. Thus, a crucial problem is how to approximate such a $E[R_s(K)]$. Here, we use $M|M|1||K$ terminal model in queuing theory to approximate $E[R_s(K)]$.

Before starting further proof, we need introduce the definition of terminal model first.

Definition 3. $M|M|1||K$ ***Terminal Model***
In the $M|M|1||K$ queue model that is shown in Fig. 2, there are K system users sitting behind their terminals. These users send requests after exponentially distributed think time Z with mean $E[Z] = 1/\lambda$ (λ, namely arrival rate). The more the users in the thinking state, the higher the effective completion rate of the thinkers. Thus, the effective arrival rate of the system is proportional to the number of thinkers. And so, in the system model, users are modelled to be infinite servers with each user is configured with its own server. The requests once are submitted by the users, only need to wait until the response. The system completes a job with mean time $E[S] = 1/\mu$ (μ, namely service rate).

Such a terminal model is an example of a closed-queuing network with a population of K customers circling between terminals and the server system. In fact, the scheduling model can be represented by a typical terminal model.

First, according to the $M|M|1||K$ terminal model, computation of $T[R(t)]$ and $E[R_s(K)]$ will be specified in Propositions 1 and 3.

Proposition 1. *The transient response time, $T[R(t)]$, can be computed over the sum of transient waiting time and transient service time.*

Proof: Based on Definition 3, each scheduling process can be modelled within a single server system, where each type of server can only serve one customer at a time. Thus, the transient response time $T[R(t)]$ at the server system equals the sum of transient waiting time $T[W(t)]$ for all customers in the queuing station and the transient service time $T[S(t)]$ for a single customer in the server, as $T[R(t)] = T[W(t)] + T[S(t)]$. Here, the arrival rate is represented by λ, the service rate by μ, and the transient queue length by q. The formula for calculating transient

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2934890, IEEE Access

response time is stated in Eq.(3).

$$\begin{aligned} T[R(t)] &= T[W(t)] + T[S(t)] \\ &= \frac{q}{\mu} + \frac{1}{\mu} = \frac{q+1}{\mu} = (q+1) \cdot E[S] \end{aligned} \tag{3}$$

Next, on the basis of Definition 3 and Proposition 1, average response time $E[R_s(K)]$ can be measured with a method known as mean-value analysis that well applies to generic queuing models, such as our scheduling model. To obtain the approximation of $E[R_s(K)]$, we need utilize the expression of system throughput based on the terminal model, which is specified in Proposition 2.

Proposition 2. *The throughput $X(K)$ of system can be formulated with average response time $E[R_s(K)]$.*

Proof: First of all, we should address the average response time at terminals $E[R_t]$ and at the server system $E[R_s(K)]$. In an infinite-server queuing station, every job has its server; thus there is no occurrence of queuing or waiting. Therefore, $E[R_t] = E[Z]$, which is independent to the number of K customers actually present. For processing the server system, however, the average response time depends on the number of K customers. To compute $E[R_s(K)]$, the average circle time should be introduced here as $E[C(K)] = E[R_t] + E[R_s(K)] = E[Z] + E[R_s(K)]$. Here, E[C(K)] expresses the mean time for a customer to go through the cycle of the think-serve once. The throughput $X[K]$ can now be represented as $K/E[C(K)]$, which is the product of the frequency with users cycle $1/E[C(K)]$ and the number of users K. Hence, combining above two expressions, we obtain Eq.(4).

$$X[K] = \frac{K}{E[C(K)]} = \frac{K}{E[Z] + E[R_s(K)]} \tag{4}$$

Additionally, the computation of average response time, $E[R_s(K)]$, is defined in Proposition 3.

Proposition 3. *The average response time, $E[R_s(K)]$, obtains its approximating formulation in the condition of large-scale K and Proposition 2.*

Proof: From Eq.(4) of Proposition 2, we derive the response time law shown as:

$$E[R_s(K)] = \frac{K}{X[K]} - E[Z] \tag{5}$$

As we know, the throughput $X[K]$ equals the product of the server-busy probability and the service rate of system, as $X[K] = (1 - p_0) \cdot \mu = (1 - p_0)/E[S]$, in which p_0 is the probability of server-idle. However, if we change the value of K, p_0 also changes; thus, p_0 should be considered a function of K. Therefore, Eq.(5) is converted to the following expression:

$$E[R_s(K)] = \frac{K \cdot E[S]}{1 - p_0(K)} - E[Z] \tag{6}$$

The aim is to model scheduling process with a massive number of tasks in the system and so the value of K is set on a large scale. For a large K, the idle fraction is very small so that the denominator $1 - p_0(K)$ will approach 1. Consequently, based on Eq.(6), the approximation of $E[R_s(K)]$ is obtained as follows:

$$E[R_s(K)]K \cdot E[S] - E[Z] \tag{7}$$

Finally, we have both the transient response time $T[R(t)]$ in Lemma 1 and the average response time $E[R_s(K)]$ in Lemma 3. Hence, the decision function $DF_{TDS}(t)$ of TDS algorithm, which is noted in Eq.(2), can be reformulated as:

$$DF_{TDS}(t) = \frac{E[R_s(K)]}{T[R]} = \frac{K \cdot E[S] - E[Z]}{(q+1) \cdot E(S)} \tag{8}$$

In Eq.(8), all parameters are specified with constant values; thus, the value of $DF_{TDS}(t)$ is computed in real time and used for scheduling decision support.

4.3 Algorithm Analysis: QDS and TDS

To effectively utilize QDS and TDS algorithms, their performance need be analysed theoretically. Suppose the heterogeneous system environment is considered with two main factors: the varying task arrivals (i.e., using a varying $lambda$) and server capability (i.e., using a varying μ).

Regarding Definitions 1 and 2, the crucial sections of $p(Q)$ and $p(T)$ are $T[Q(t)]$ and $T[R(t)]$, respectively, which are two dynamic factors affecting the value of decision function. Increased values of these factors, $T[Q(t)]$ and $T[R(t)]$, represent the growing waiting queue and response time, which lead to the decrease of their corresponding decision functions. Thus, both algorithms will reduce the dispatching tasks to a related server.

Under a heterogeneous system, on one hand, when only task arrivals (i.e., λ) are changing, $T[R(t)]$ is linearly proportional to $T[Q(t)]$ because of the unchanging server capability (i.e., μ). Both $T[Q(t)]$ and $T[R(t)]$ vary with the changing λ. Hence, two algorithms should have close performance, but $p(Q)$ should be a bit better due to its efficient calculation by only observing the waiting queue of each server; nevertheless, $p(T)$ need calculate both average and transient response time. On the other hand, when only server capability (i.e., μ) is varying, the service time span of each task is no longer the same owing to the varying μ. Larger μ generates a faster service time (i.e., $1/\mu$), on the contrary, longer service time. As a result, if a queue has less waiting tasks but in a lower-rate server, it is possible that there is relatively longer response time in this server; however, another server with larger μ might provide faster response even it has a longer waiting queue. In this situation, it is not accurate to decide an allocation based on the transient queue length of each server, namely algorithm $p(Q)$. Hence, algorithm $p(T)$ should be better in such a condition.

Furthermore, according to Definition 1, $p(Q)$ is designed on calculation with a transient queue length of a server. Similarly, from Proposition 3, $p(T)$ generates a decision value on the basis of a ratio of average response time $E[R_s(K)]$ and transient response time $T[R]$, in which $E[R_s(K)]$ is calculated from a set of constant system parameters and $T[R]$ can be obtained from a transient queue length of the server. Thus, both $p(Q)$ and $p(T)$ generate their decision results only depending on the dynamic transient queue

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2934890, IEEE Access

length that can be observed with an one-time operation. Hence, two algorithms have the same time complexity $O(1)$. For this reason, the HDSS does not consider the difference of time complexity between two algorithms when design decision support algorithm DSF in Algorithm 1.

According to above analysis, to adaptive a heterogeneous system, both two conditions (i.e., varying λ and μ) must be considered by jointly using two algorithms as a hybrid scheme that will be detailed in next section.

5 Hybrid Dynamic Scheduling Scheme

The VEC system usually has heterogeneous system conditions including changing task arrivals and service capability due to the mobility of vehicles. During rush hours, vehicles move slower so that both task requests and vehicular resources remain relatively stable distribution. However, on a clear and open road, vehicles move fast, which causes a frequently changing distribution of task requests and service resources. Hence, such situation brings forward higher demand to scheduling process. For the reason, this section aims to design a QoS-aware hybrid dynamic scheduling scheme (HDSS) by aggregating multiple scheduling algorithms to adapt heterogeneous system conditions.

HDSS is a scheme with multiple scheduling algorithms, which consists of two modules: a Decision-support Module (DSM) and a Task Scheduling Module (TSM).

In HDSS, DSM is represented by a decision support function, $DSF(v_1, v_2)$, which is used to decide the key factor that affects system environment by comparing two input values and then output the decision result v_{out}. The input values of DSF are generated from another function $Var(r)$ that is defined to measure the dispersion of a given probability distribution, such as a distribution of task arrival rates or a distribution of service rates. A larger value of $Var(r)$ denotes higher degree of dispersion; rather, smaller $Var(r)$, lower dispersion. In the scheme, $Var(r)$ is conducted on calculation of coefficient of variation (CV), which will be introduced in scheme details. Based on the results of $DSF(v_1, v_2)$, DSM selects an optimal algorithm (either QDS or TDS); thereafter, TSM conducts task scheduling with the selected algorithm. A complete description of HDSS is shown in Algorithm 1.

As defined in Algorithm 1, in Step 1, HDSS first initializes the system by specifying conditions of job streams and server systems. Step 2 evaluates system environments by calculating CVs (i.e., CV_λ and CV_μ) based on two key system factors: arrival rate λ and service rate μ. Higher value of CV represents a dominant position of this factor in the system. Thereafter, in Step 3, DSM applies the decision support function $DSF(v_1, v_2)$ to determine the selection of scheduling algorithms. According to the analysis in Section 4.3, the scheme is designed as: if the variation of arrival rates affects the system relatively more (i.e., $CV_\lambda > CV_\mu$), QDS algorithm will be selected; conversely, if service rate has more influence on the system (i.e., $CV_\lambda < CV_\mu$), the scheme will adopt TDS algorithm. Finally, in Step 4, TSM implements scheduling operation based on the decision of DSM. The reason of such assumption will be verified in the following performance evaluation section.

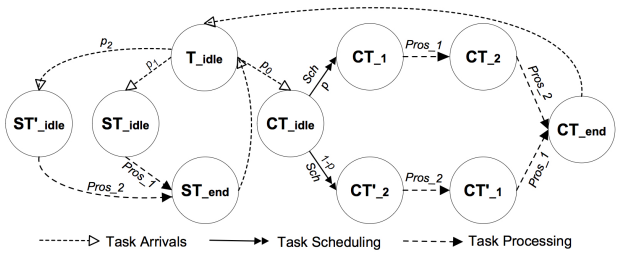

Fig. 3. Scheduling Model Framework with Different Types of Tasks

6 PEPA-based Scheduling Models

In the section, a general scheduling model is built with a novel formal method – PEPA (Performance Evaluation Process Algebra), which is a high-level model specification language, defined by Hillston [31]. In contrast to classical process algebras, each activity of PEPA is defined with a random duration following exponential distribution. Due to the memoryless property of exponential distribution, the stochastic process indicated by PEPA has the Markov property. Hence the underlying stochastic process is a CTMC.

6.1 PEPA Syntax

PEPA is used as our modelling tool owing to its superiority in: *Formality*, PEPA language has a structured operational semantics and provides a formal interpretation for all expressions; *Compositionality*, the compositional nature provides the ability to model a system as the interaction of subsystems; *Abstraction*, PEPA is able to construct complex models from detailed system components, disregarding the details when it is appropriate to do so. The syntax of PEPA is given as:

$$P ::= (a, \lambda).P \ \mid \ P + Q \ \mid \ P\langle L\rangle Q \ \mid \ P \,|\, Q \ \mid \ A,$$

which involves four combinators: **Prefix**: $(\alpha, r).P$: Prefix is a basic mechanism describing system behaviours; **Choice**: $P + Q$: The component $P + Q$ represents a competition between two components; **Cooperation**: $P \underset{L}{\bowtie} Q$: The cooperation combinator describes the synchronisation of P and Q over the activities in the cooperation set L; **Hiding**: P/L: Hiding makes the activities whose action types are in L invisible for an external observer; **Constant**: $A \stackrel{\text{def}}{=} P$: The last part of this statement $P ::= A$ identifies a component P with a constant value A.

6.2 Scheduling Model Framework

To represent scheduling process, we first describe a system framework to build a general scheduling process that is depicted in Fig. 3. The framework holds all types of queuing tasks in an initial state T_{idle}, and also remains two assumptions: system provides two types of services noted as $Pros_1$ and $Pros_2$; and, three types of tasks consisting of a Single-type Task (ST) requiring $Pros_1$ service only, another Single-type Task (ST') requiring $Pros_2$ service only and a Complex-type Task (CT) requiring both $Pros_1$ and $Pros_2$.

All Single-type Tasks (ST and ST') change their states from ST_{idle} and ST'_{idle} to ST_{end} by conducting their corresponding services (either $Pros_1$ or $Pros_2$), and then switch

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2934890, IEEE Access

Algorithm 1: Hybrid Dynamic Scheduling Scheme – HDSS

Input: $J\ (a\ set\ of\ jobs);\ S\ (a\ set\ of\ servers).$
Output: *Dynamic Scheduling Operation*
Step 1: System Initialization ;
Specify $\forall j_k \in J,\ j_k =< k, v, \lambda, p >,\ (k = 1, 2, \cdots, n)$, and $\forall s_i \in S,\ s_i =< i, v, \mu >,\ (i = 1, 2, \cdots, m)$;
Step 2: System Environment Evaluation Based on Coefficient of Variation (CV) ;
while *System Running* **do**
  if *System Updated* **then**
    Calculate Standard Deviation (σ) based on λ and μ: $\sigma_\lambda = \sqrt{\frac{1}{n}\sum_{k=1}^{n}(\lambda_k - \overline{\lambda})^2};\ \sigma_\mu = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(\mu_i - \overline{\mu})^2}$;
    Update Coefficient of Variation (CV): $CV_\lambda = \frac{\sigma_\lambda}{\overline{\lambda}} \times 100\%;\ CV_\mu = \frac{\sigma_\mu}{\overline{\mu}} \times 100\%$;
  end
end
Step 3: Decision Support with $DSF(v_1, v_2)$;
for *each* j_k *in queue,* $j_k \in J$ **do**
  if CV_λ *and* CV_μ *Updated* **then**

$$\left.\begin{aligned} v_1 = Var(\lambda) = CV_\lambda \\ v_2 = Var(\mu) = CV_\mu \end{aligned}\right\} \implies DSF(CV_\lambda, CV_\mu) = v_{out} = \begin{cases} 1, & \text{if } CV_\lambda > CV_\mu; \\ 0, & \text{if } CV_\lambda < CV_\mu. \end{cases}$$

  end
  Step 4: Scheduling Algorithm Selection ;
  if $v_{out} = 1$ **then**
    Use Scheduling Algorithm QDS: $DF_{QDS}(t) = C \cdot \frac{1}{T[Q(t)]}$
  else
    Use Scheduling Algorithm TDS: $DF_{TDS}(t) = \frac{E[R_s(K)]}{T[R]} = \frac{K \cdot E[S] - E[Z]}{(q+1) \cdot E(S)}$
  end
end

back to their initial state ST_{idle}. Nevertheless, the Complex-type Tasks (CT) need conduct both operations (i.e., $Pros_1$ and $Pros_2$) in any order, and the states change from CT_{idle} to CT_{end} via two possible routes either $CT_1 \rightarrow CT_2$ or $CT_2' \rightarrow CT_1'$. The selection of routes depends on a scheduling model that will be illustrated later.

6.3 PEPA-based Scheduling Models

First, based on the model framework, each type of tasks is initially defined with a probability $p \in < p_0, p_1, p_2 >$ representing each branch of task generating activity. In the modelling, such probabilities are usually specified with the statistic figures or prediction of a real-world system. In other words, the probabilities can be considered the proportion of each type of tasks. Therefore, the actual rates of task generating activities can be adjusted by adopting these probabilities. Thus, the system which will perform a task-generating action of type $send$ at rate λ, and then with probability p_0, behave as component CT, and with probability p_1 and p_2, behave as components ST and ST', respectively, will be defined as a PEPA component TSK enabling three types of $send$ activities:

$$TSK \stackrel{\text{def}}{=} (send,\ p_0 \cdot \lambda).CT_{idle} + (send,\ p_1 \cdot \lambda).ST_{idle} + (send,\ p_2 \cdot \lambda).ST'_{idle}.$$

Second, for each type of tasks, their activities can be modelled as follows.

Modelling of Single-type Task Component:

$$ST_{idle} \stackrel{\text{def}}{=} (pros_1,\ \mu_1).ST_{end};$$
$$ST'_{idle} \stackrel{\text{def}}{=} (pros_2,\ \mu_2).ST_{end}.$$

Each single-type task comes from an idle state (either ST_{idle} or ST'_{idle}) to the end state ST_{end} after its corresponding service process $pros_1$ with rate μ_1 or $pros_2$ with rate μ_2.

Modelling of Complex-type Task Component:

$$CT_{idle} \stackrel{\text{def}}{=} (pros_1,\ \mu_1).(pros_2,\ \mu_2).CT_{end} + (pros_2,\ \mu_2).(pros_1,\ \mu_1).CT_{end}$$

Each complex-type task completes its operations in the order of action flows: either $pros_1 \rightarrow pros_2$ or $pros_2 \rightarrow pros_1$.

Next, in PEPA models, server components need be defined independently, which are cooperatively used with task components. These server components can be defined as:

$$VFC_{server\_1} \stackrel{\text{def}}{=} (pros_1,\ \mu_1).VFC_{server\_1};$$
$$VFC_{server\_2} \stackrel{\text{def}}{=} (pros_2,\ \mu_2).VFC_{server\_2}.$$

Third, in the model of complex-type task component, a scheduling algorithm need be specified to decide which server should be selected first for the current task on the basis of system conditions. To address the issue, a hybrid dynamic scheduler component HDS is defined in the model, as follows:

$$HDS \stackrel{\text{def}}{=} (Sch\_QD,\ \Gamma_{QD}).HDS + (Sch\_TD,\ \Gamma_{TD}).HDS.$$

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2934890, IEEE Access

In above statement, Sch_QD and Sch_TD represent two types of scheduling operations based on QDS and TDS algorithms respectively with their corresponding action rates Γ_{QD} and Γ_{TD}. To implement dynamic scheduling algorithms, we need use their related decision functions as control factors (CF, noted as f) of action rates Γ_{QD} and Γ_{TD}. Thus, the HDS statement is altered to a new format:

$$\begin{aligned} HDS \stackrel{\text{def}}{=} & (Sch\_QD,\ f_{QD} \cdot \Gamma_{QD}).HDS \\ & + (Sch\_TD,\ f_{TD} \cdot \Gamma_{TD}).HDS, \end{aligned}$$

where $f_{QD} = DF_{QDS}(t)$ and $f_{TD} = DF_{TDS}(t)$.

To model a scheduling operation, the scheduler component HDS need cooperate with the CT component by updating the preceding statement as:

$$\begin{aligned} & CT_{idle} \\ \stackrel{\text{def}}{=} & (Sch\_QD,\ f_{QD} \cdot \Gamma_{QD}).(pros_1,\ \mu_1).(pros_2,\ \mu_2).CT_{end} \\ & + (Sch\_QD,\ f_{QD} \cdot \Gamma_{QD}).(pros_2,\ \mu_2).(pros_1,\ \mu_1).CT_{end}, \end{aligned}$$

which adopts the QDS algorithm for scheduling; and an alternative type of CT statement is denoted as:

$$\begin{aligned} & CT_{idle} \\ \stackrel{\text{def}}{=} & (Sch\_TD,\ f_{TD} \cdot \Gamma_{TD}).(pros_1,\ \mu_1).(pros_2,\ \mu_2).CT_{end} \\ & + (Sch\_TD,\ f_{TD} \cdot \Gamma_{TD}).(pros_2,\ \mu_2).(pros_1,\ \mu_1).CT_{end}, \end{aligned}$$

which uses the TDS algorithm in scheduling process.

Finally, the whole system is represented by joining these defined components as a global system component, Sys, which is represented as:

$$\begin{aligned} Sys \stackrel{\text{def}}{=} & TSK[n] \underset{\mathrm{L}}{\bowtie} \\ & (VFC_{server\_1}[m_1] \| VFC_{server\_2}[m_2] \ \| HDS[m_0]) \\ & L = \{pros1,\ pros2,\ Sch\_QD,\ Sch\_TD\}, \end{aligned}$$

in which TSK cooperatively executes with other components by specifying their shared actions in a set L, and $< n, m_0, m_1, m_2 >$ denote the number of instances of a corresponding component.

7 Performance Evaluation

To generate performance measuring, a novel fluid analysis approach is used on a labelled multi-transition system derived from a PEPA model.

7.1 Fluid Flow Approximation

On the structured operational semantical rules [31], a PEPA model can be regarded as a corresponding multi-transition system. For each local derivative P or Q, it can either perform an activity l that is $P \xrightarrow{l}$ or be obtained by performing l that is $\xrightarrow{l} Q$. Here, P is called a pre local derivative of l; and Q is a $post$ local derivative of l. The local derivative set that is derived from P after activity l is named the $post$ set of l from P, which is denoted by $post(P, l) = \{Q | P \xrightarrow{l} Q\}$. Each $l^{P \to Q}$ or l^{ω} is called a labelled activity. Hence, the set of all labelled activities is denoted by $\mathcal{A}_{label}$.

In the labelled multi-transition system, the transient rates ($r_l^{P \to Q}$) of a PEPA model are represented by a transient rate function [32], $f(\mathbf{x}, l)$ that means the transient rate function of action l in state $\mathbf{x}$. If l is an individual activity (e.g., $send$ action of PEPA model), for each $P \xrightarrow{(l, r^{P \to Q})} Q$, we have:

$$f(\mathbf{x}, l^{P \to Q}) = \mathbf{x}[P] \cdot r_l^{P \to Q} \ ,$$

moreover, if l is a shared action (e.g., $pros$ action) with the pre-set of l, $pre(l) = (P_1, P_2, \cdots, P_k)$, then for each $(Q_1, Q_2, \cdots, Q_k)$ in $post(P_1, l) \times post(P_2, l) \times \cdots \times post(P_k, l)$, the transition rate function of labelled activity l_ω $(\omega = P_1 \to Q_1, P_2 \to Q_2, \cdots, P_k \to Q_k)$ in state $\mathbf{x}$ is:

$$f(\mathbf{x}, l^{\omega}) = \left(\prod_{i=1}^{k} \frac{r_l^{P_i \to Q_i}}{r_l(P_i)} \right) \min_{i \in \{1, \cdots, k\}} \{ r_l(\mathbf{x}, P_i) \}.$$

With the transient rate function, a fluid analysis can be conducted by obtaining a set of ordinary differential equations (ODEs) derived on the basis of a PEPA model rather than directly solving its underlying CTMC.

The simulation method has drawback of high computational cost and low efficiency in real-time performance measuring or large-scale performance analysis. Thus, a novel approach to conduct measures and analysis will be adopted in this work, which is called *Fluid Flow Approximation* [33], generating a continuous state space approximation with evolution governed by a set of ODEs.

According to the labelled multi-transition system, the transition rate $f(\mathbf{x}, l)$ determines the transition intensity from state $\mathbf{x}$ to state $\mathbf{x} + l$. The system state space is inherently discrete with entries in the numerical vector form, which is incremented or decremented in a single step with a change in a system state. As explained in [33], with a large number of components, these steps can be considered tiny movements between states. Thus, we can approximate these discrete but tiny steps to be continuous, rather than happening in discontinuous skips. For the evolution of numerical state vectors, denoting the state at time t as $\mathbf{x}(t)$, and within a very short time interval δt, the change to the state vector $\mathbf{x}(t)$ can be represented as:

$$\mathbf{x}(\cdot, t + \delta t) - \mathbf{x}(\cdot, t) = \delta t \sum_{l \in \mathcal{A}_{label}} l f(\mathbf{x}(\cdot, t), l) \ ,$$

Taking a limit ($\delta \to 0$) after a division by δt, a set of ODEs can be obtained by:

$$\frac{d\mathbf{x}}{dt} = \sum_{l \in \mathcal{A}_{label}} l f(\mathbf{x}, l) \ .$$

7.2 Fluid Analysis Results

To evaluate performance of scheduling algorithms, a set of parameters need be initialized under the predefined PEPA model. As mentioned in Section 7.1, a large number of task instances is required to conduct fluid analysis. Thus, the total number of tasks is set to 4000 consisting of equal number of single-service tasks (1000 STs and 1000 ST's) and multi-service tasks (2000 CTs), and each other component has unique instance. Each incoming task will be queued for scheduling to its destination server based on the selected dynamic algorithm. To facilitate the modelling, this task scheduling process is modelled with a "stop-wait" mode in the queue. Related rate conditions are given in Table 2. Each type of task arrivals in system with a equal

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2934890, IEEE Access

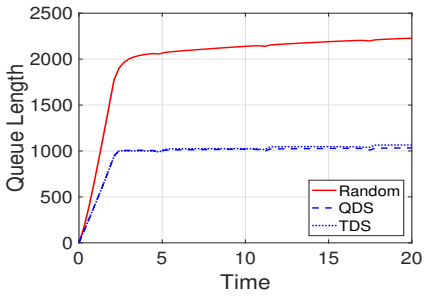

Fig. 4. Comparing Three Algorithms with Unstable Arrival Rates

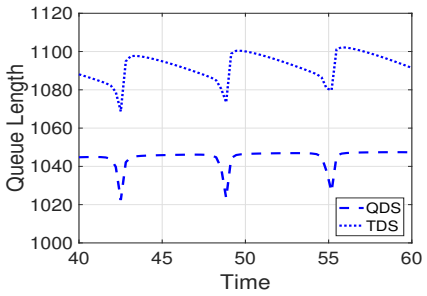

Fig. 5. Comparing QDS and TDS with Unstable Arrival Rates

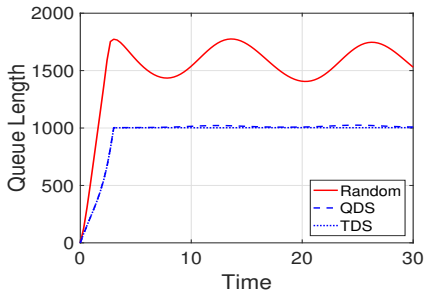

Fig. 6. Comparing Three Algorithms with Unstable Service Rates

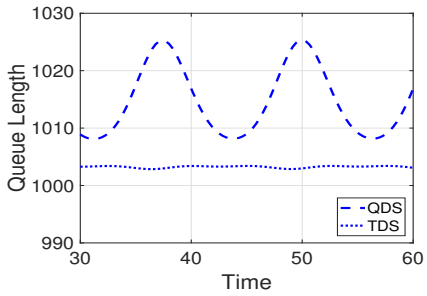

Fig. 7. Comparing QDS and TDS with Unstable Service Rates

rate 400 (Tasks per Time unit), and two processing actions have their rates to be 120 and 80, respectively. To model the variation of arrival rates and service rates, different trigonometric functions (e.g., sin and cos) are used to govern the fluctuation of rate values.

TABLE 2
Model Parameter Setting

Arrival Rate	Value	Service Rate	Value
λ_{ST}	400	μ_1	120
$\lambda_{ST'}$	400	μ_2	80
λ_{CT}	400		

In measurements, we initially compare two dynamic scheduling algorithms with the random scheduling algorithm under two types of system conditions: task-arrival rate (λ) varying only and server-service rate (μ) varying only. For the first type of condition, Figs. 4-5 depict the changing amount of waiting tasks of three algorithms with varying arrival rates, in which both dynamic algorithms can efficiently conduct scheduling process than the random algorithm based on Fig. 4, and QDS has a bit better performance than TDS due to its relatively stable growth and lower position in Fig. 5.

Conversely, under the second type of condition, TDS algorithm has much better performance as shown in Figs. 6-7 representing the number of waiting tasks with varying service rate. It is worth to mention that the unstable service rate easily causes serious impact on the waiting queue, which means the inferior QoS of scheduling especially for the random algorithm in Fig. 6. However, TDS algorithm, comparing with QDS in Fig. 7, shows its well performance in keeping the waiting queue smoother and shorter.

When both λ and μ keep varying in the system, the selection of dynamic scheduling algorithm depends on the primary factor, either λ or μ, which affects system stability more. This will be determined by the proposed decision support function $DSF(v_1, v_2)$ in Section 5.

According to the measurements, it is clear that $DSF(v_1, v_2)$ defined in Algorithm 1 is reasonable. When $CV_\lambda > CV_\mu$ (i.e., conditions in Figs. 4-5) representing that the varying arrival rate dominates the change of system, QDS algorithm need be selected because of its efficient and superior performance; but when $CV_\lambda < CV_\mu$ (i.e., conditions in Figs. 6-7) representing that the system is affected more by the varying service rate, TDS algorithm should be selected as the optimal scheduling solution by the decision function. In order to demonstrate the performance of the hybrid scheme HDSS, further analysis will be conducted by comparing HDSS with another hybrid scheduling scheme in Section 7.3.

7.3 Performance Comparison

To evaluate the performance of HDSS, it is necessary to compare it with a similar hybrid algorithm that is a randomized Join-the-Shortest-Queue (R-JSQ) scheme in Ref. [30]. According to Ref. [30], JSQ algorithm benefits efficient scheduling operation but needs to observe all active servers for their queueing situation, which increase costs especially in large-scale systems. Therefore, a hybrid scheme, i.e. R-JSQ, is proposed by scheduling tasks with two steps: R-JSQ first selects two servers uniformly at random from a server set with a capacity fitting to the given task set; thereafter, it adopts JSQ algorithm and routes the task to the server that has the lease number of unfinished tasks among the two selected servers. R-JSQ is a hybrid scheme of Random and JSQ algorithms.

To achieve fair comparison, the same model framework is designed in the experiment, which has two steps: First,

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2934890, IEEE Access

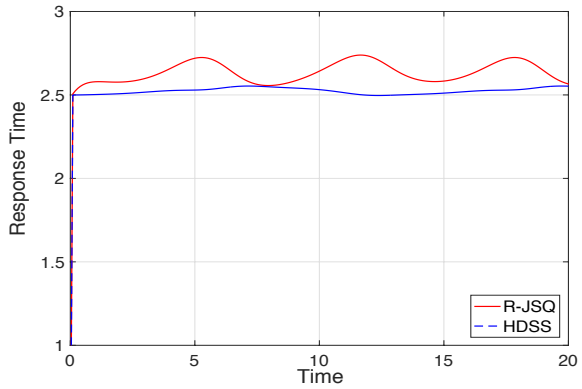

Fig. 8. Performance Comparison between HDSS and R-JSQ

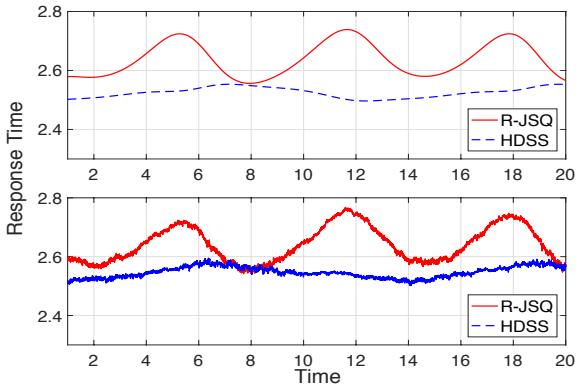

Fig. 9. Validation of Fluid Analysis through Simulation

classify servers in different groups based on capacity level; Second, use either R-JSQ and HDSS as the scheduling principle, respectively, for two selected servers from the same group sorted in the first step. Furthermore, in the experiment, a heterogeneous system environment is generated by modelling arrival streams and server capacity in exponential distribution with sine-waved means. Evaluation is first conduced through the fluid-flow analysis based on their PEPA models. As the the system condition is set with a greater vibration rate of server capacity, HDSS adopts TDS algorithm in the scheduling stage, which aims to effectively compare R-JSQ and HDSS. As shown in Fig. 8, HDSS generates faster and more stable response than R-JSQ, which means a superior performance in the condition of heterogeneous server capacity. To validate the fluid-flow analysis results, discrete event simulation is also implemented with the same model framework. Fig. 9 depicts the fact that the fluid-flow analysis coincides with the simulation results.

8 Conclusion

This paper proposed a hybrid dynamic scheduling scheme towards heterogeneous VEC systems. The HDSS implements a dual-scheduling framework to adapt different system conditions based on two embedded dynamic scheduling algorithms (QDS and TDS).The design of HDSS aims to solve the drawback of traditional Join-the-Shortest-Queue algorithm especially in a condition of unstable system capacity. The proposed response time-based dynamic scheduling algorithm (TDS) can efficiently adjust scheduling in terms of heterogeneous server capacity. Furthermore, HDSS is defined as a hybrid scheme by integrating both TDS and QDS, which can adapt heterogeneous conditions of servers and arrivals. A decision support function is also designed to select the optimal scheduling algorithm based on the current condition of the system. According to the performance experiments, it is reasonable that HDSS well suits a heterogeneous system environment especially under the condition of varying server capacity because of the proposed novel Time-based Dynamic Scheduling (TDS) algorithm. In addition, the HDSS can also efficiently switch to another scheduling algorithm with the change of system conditions in order to gain superior performance.

In the future, cost-efficient scheduling scheme will be concerned more in large-scale and hybrid-architecture systems (e.g., mixed cloud-edge systems). Thus, our future works aims to explore the scheduling scheme by considering more complex system scenarios. Moreover, further experiments and benchmark with other newly designed scheduling algorithms need be conducted to enhance its performance evaluation.

Acknowledgments

The authors acknowledge the financial support by the National Natural Science Foundation of China (NSFC) under Grants 61702233 and 61472343, the Natural Science Foundation of Jiangsu Province (China) under Grant BK20160543, and State Key Laboratory of Software Development Environment under Grant SKLSDE-2017KF-03.

References

[1] E. Lee, E. K. Lee, M. Gerla and S. Y. Oh, *Vehicular Cloud Networking: Architecture and Design Principles*, IEEE Communications Magazine, Vol.52, No.2, pp.148-155, 2014.

[2] M. Whaiduzzaman, M. Sookhak, A. Gani and R. Buyya, *A Survey on Vehicular Cloud Computing*, Journal of Network and Computer Application, Vol.40, No.1, pp.325-344, 2014.

[3] M. Agiwal, A. Roy and N. Saxena, *Next Generation 5G Wireless Networks: A Comprehensive Survey*, IEEE Communications Surveys & Tutorials, Vol.18, No.3, pp.1617-1655, 2016.

[4] W. Kuo, Y. Tung and S. Fang, *A Node Management Scheme for R2V Connections in RSU-supported Vehicular Ad-hoc Networks*, in Proceedings of 2013 ICNC, pp.768-772, San Diego, CA, USA, 2013.

[5] N. Fernando, S. W. Loke and W. Rahayu, *Mobile Cloud Computing: A Survey*, Future Generation Computer System, Vol.29, No.1, pp.84-106, 2013.

[6] P. Ghazizadeh, R. Florin, A. G. Zadeh et al. *Reasoning About Mean Time to Failure in Vehicular Clouds*, IEEE Transactions on Intelligent Transportation Systems, Vol.17, No.3, pp.751-761, 2016.

[7] R. Yu, G. Xue, V.T. Kilari, X. Zhang, *The Fog of Things Paradigm: Road toward On-Demand Internet of Things*, IEEE Communications Magazine, Vol. 56, No. 9, pp. 48-54, 2018.

[8] R. Yu, G. Xue and X. Zhang, *Application Provisioning in FOG Computing-enabled Internet-of-Things: A Network Perspective*, 2018 IEEE Conference on Computer Communications (INFOCOM'18), Honolulu, HI, 2018, pp. 783-791.

[9] F. Bonomi, *Connected Vehicles, the Internet of Things, and Fog Computing*, in Proceedings of VANET, pp. 13-15, Las Vegas, USA, 2011.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2934890, IEEE Access

[10] X. Hou, M. Chen, D. Wu, D. Jin and S. Chen, *Vehicular Fog Computing: A Viewpoint of Vehicles as the Infrastructures*, IEEE Transactions on Vehicular Technology, Vol.65, No.6, pp.3860-3873, 2016.

[11] L. Pu, X. Chen, G. Mao, Q. Xie and J. Xu, *Chimera: An Energy-Efficient and Deadline-Aware Hybrid Edge Computing Framework for Vehicular Crowdsensing Applications*, IEEE Internet of Things Journal, vol. 6, no. 1, pp. 84-99, Feb. 2019.

[12] S. S. Shah, M. Ali, A. W. Malik, M. A. Khan and S. D. Ravana, *vFog: A Vehicle-Assisted Computing Framework for Delay-Sensitive Applications in Smart Cities*, IEEE Access, vol. 7, pp. 34900-34909, 2019.

[13] Z. Li, J. Ge, H. Yang, L. Huang, H. Hu, H. Hu and B. Luo, *A Security and Cost Aware Scheduling Algorithm for Heterogeneous Tasks of Scientific Workflow in Clouds*, Future Generation Computer Systems, Vol.65, pp.140-152, 2016.

[14] C. H. Chen, *Task Scheduling for Maximizing Performance and Reliability Considering Fault Recovery in Heterogeneous Distributed Systems*, IEEE Transactions on Parallel and Distributed Systems, Vol. 27, No. 2, pp.521-532, 2016.

[15] T. Liu, F. Chen, Y. Ma and Y. Xie, *An Energy-efficient Task Scheduling for Mobile Devices Based on Cloud Assistant*, Future Generation Computer Systems, Vol. 61, pp. 1-12, 2016.

[16] J. Jiang, S. Ma, B. Li and B. Li, *Symbiosis: Network-aware Task Scheduling in Data-parallel Frameworks*, in Proceedings of 2016 IEEE INFOCOM, pp.766-774, San Francisco, CA, 2016.

[17] V. Gupta, M. H. Balter, K. Sigman and W. Whitt, *Analysis of join-the-shortest-queue routing for web server farms*, Performance Evaluation, Vol. 64, Iss. 9, pp. 1062-1081, 2007.

[18] R. V. Lopes and D. Menasc, *A Taxonomy of Job Scheduling on Distributed Computing Systems*, IEEE Transactions on Parallel and Distributed Systems, Vol. 27, No. 12, pp. 3412-3428, 2016.

[19] S. T. Maguluri, R. Srikant and L. Ying, *Stochastic Models of Load Balancing and Scheduling in Cloud Computing Clusters*, in Proceedings of 2012 IEEE INFOCOM, pp. 702-710, Orlando, FL, 2012.

[20] C. W. Tsai and J. J. P. C. Rodrigues, *Metaheuristic Scheduling for Cloud: A Survey*, IEEE Systems Journal, Vol. 8, No. 1, pp. 279-291, 2014.

[21] J. Feng, Z. Liu, C. Wu and Y. Ji, *AVE: Autonomous Vehicular Edge Computing Framework with ACO-Based Scheduling*, IEEE Transactions on Vehicular Technology, vol. 66, no. 12, pp. 10660-10675, Dec. 2017.

[22] J. Feng, Z. Liu, C. Wu and Y. Ji, *Mobile Edge Computing for the Internet of Vehicles: Offloading Framework and Job Scheduling*, IEEE Vehicular Technology Magazine, vol. 14, no. 1, pp. 28-36, March 2019.

[23] H. Tan, Z. Han, X. Y. Li and F. C. M. Lau, *Online Job Dispatching and Scheduling in Edge-Clouds*, Proceedings of 2017 IEEE INFOCOM, pp.1557-1566, Atlanta, GA, 2016.

[24] D. Zeng, L. Gu, S. Guo, Z. Cheng and S. Yu, *Joint Optimization of Task Scheduling and Image Placement in Fog Computing Supported Software-Defined Embedded System*, IEEE Transactions on Computers, Vol. 65, No. 12, pp. 3702-3712, 2016.

[25] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana and M. Parashar, *Mobility-Aware Application Scheduling in Fog Computing*, IEEE Cloud Computing, Vol. 4, No. 2, pp. 26-35, 2017.

[26] J. Guo, Z. Song, Y. Cui, Z. Liu and Y. Ji, *Energy-Efficient Resource Allocation for Multi-User Mobile Edge Computing*, GLOBECOM 2017 - 2017 IEEE Global Communications Conference, Singapore, 2017, pp. 1-7.

[27] T. X. Tran and D. Pompili, *Joint Task Offloading and Resource Allocation for Multi-Server Mobile-Edge Computing Networks*, IEEE Transactions on Vehicular Technology, vol. 68, no. 1, pp. 856-868, Jan. 2019.

[28] J. Du, F. R. Yu, X. Chu, J. Feng and G. Lu, *Computation Offloading and Resource Allocation in Vehicular Networks Based on Dual-Side Cost Minimization*, in IEEE Transactions on Vehicular Technology, vol. 68, no. 2, pp. 1079-1092, Feb. 2019.

[29] A. Mahmood, B. Butler and B. Jennings, *Towards Efficient Network Resource Management in SDN-Based Heterogeneous Vehicular Networks*, 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), Tokyo, 2018, pp. 813-814.

[30] A. Mukhopadhyay, R. R. Mazumdar, *Analysis of Randomized Join-the-Shortest-Queue (JSQ) Schemes in Large Heterogeneous Processor-Sharing Systems*, IEEE Transactions on Control of Network Systems, Vol. 3, No. 2, 2016.

[31] J. Hillston, *A compositional approach to performance modelling*, Cambridge University Press, 1996.

[32] J. Ding and J. Hillston, *Numerically Representing Stochastic Process Algebra Models*, The Computer Journal, Vol.55, No.11, pp.1383-1397, 2012.

[33] J. Hillston, *Fluid Flow Approximation of PEPA Models*, in Proceedings of the Second International Conference on the Quantitative Evaluation of Systems, pp.33-43, Washington DC, USA, 2005.

XIAO CHEN received the M.Sc. and Ph.D. degrees in computing science from Newcastle University in 2009 and 2013, respectively. He is currently a research fellow at School of Informatics in the University of Edinburgh. His research interests include formal performance and optimization for large-scale systems, e.g., IoT systems, smart systems, cloud/fog systems, and Blockchain-based applications.

NIGEL THOMAS was awarded a PhD in 1997 and an MSc in 1991, both from the University of Newcastle upon Tyne. He joined the School of Computing Science at Newcastle University in January 2004 from the University of Durham, where I had been a lecturer since 1998. I was promoted to reader in 2009. My research interests lie in scalable performance analysis for networks, distributed algorithms, Cloud and IoT using queueing theory, stochastic process algebra and simulation.

TIANMING ZHAN received the Ph.D. degree in Pattern Recognition and Intelligence System from Nanjing University of Science and Technology (NUST), Nanjing, Jiangsu, China, in 2013. He is currently an Associate Professor at the School of Information and Engineering of Nanjing Audit University (NAU). His current research interests include image processing and data analysis.

JIE DING received the B.S. degree in mathematical education from Yangzhou University, Yangzhou, China, in 2001, the M.S. degree in mathematical statistics from Southeast University, Nanjing, China, in 2004, and the Ph.D. degree in communication from The Edinburgh University, Edinburgh, U.K., in 2010. He is currently a professor of Shanghai Maritime University. His research interests include performance modelling for communication and computer systems.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/.